Copy in Python – Python Deep Copy and Shallow Copy

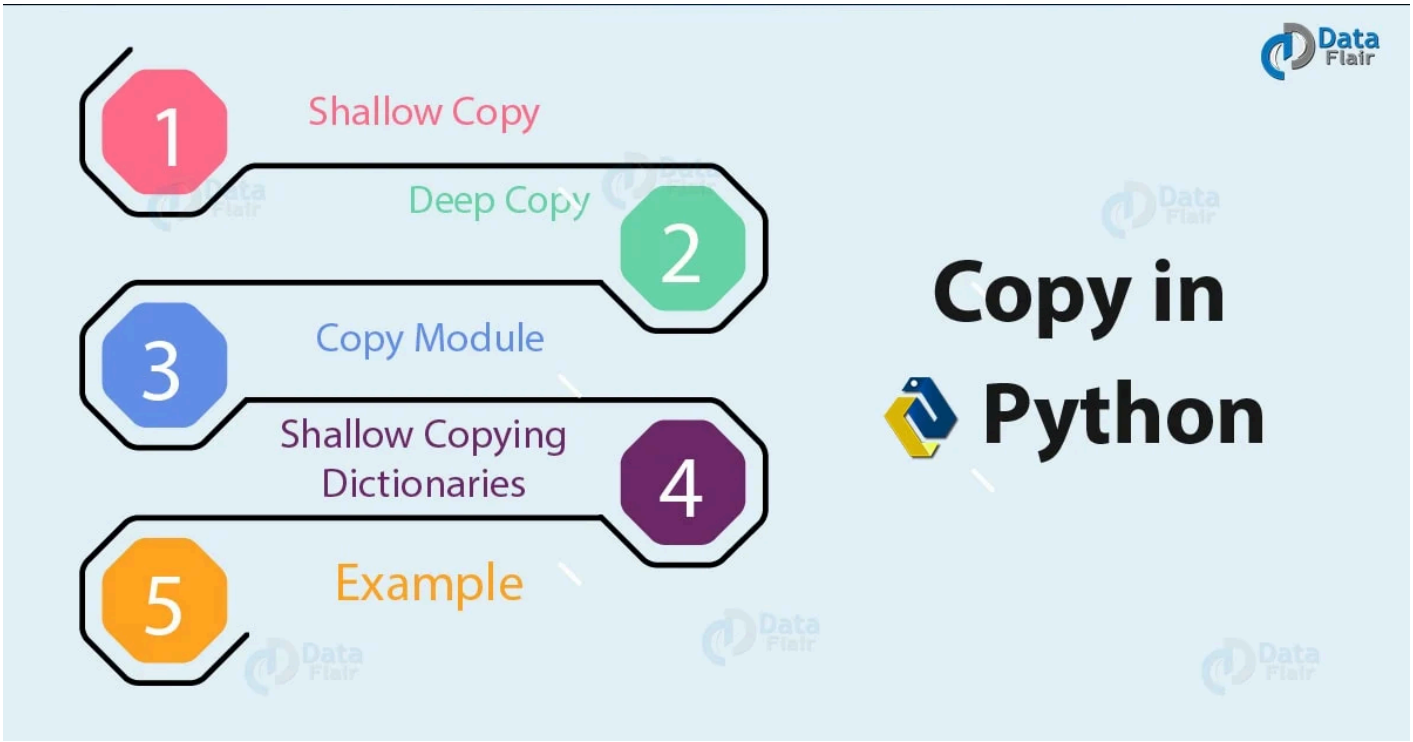
BY DATAFLAIR TEAM

Master Python with 70+ Hands-on Projects and Get Job-ready - Learn Python

Today, in this Python Tutorial, we will see the copy in Python. Moreover, we will define Deep Copy and Shallow Copy in Python.

Also, we will see a difference between Shallow Copy vs Deep Copy. Along with this, we will discuss Copy Module and Shallow Copying Dictionaries.

So, let's start Copy in Python.



Copy in Python – Shallow Copy and Deep Copy

What is Python Copy Module?

When we perform an assignment in Python, it does not copy the object we assign. All it does is create bindings between a target and an object.

But sometimes, we may need to change one Python copy without changing the other for a mutable collection.

In this Python Copy tutorial, we discuss the module copy in Python. It has the following members-

- copy.copy(x)

This returns a shallow copy of x.

- copy.deepcopy(x)

This returns a deep copy of x.

- exception copy.error

This is the exception it raises for module-specific errors.

Before we can begin explaining shallow copy and deep copy, we think it is necessary to tell you these concepts apply to compound objects only- those that hold other objects like lists or class instances.

Python Deep Copy

When a deep copy in Python creates a new object, it inserts into the new object copies of the objects in the original object. In other words, it copies an object into another.

This means any changes we make to the copy do not reflect in the original.

1. Deep Copy Example

Let's try implementing this in Python. We use the deepcopy() function.

```
>>> import copy
>>> list1=[1,3,[7,4],6]
>>> list2=copy.deepcopy(list1) #Making a deep copy
>>> list1
```

Output

```
[1, 3, [7, 4], 6]
```

```
>>> list2
```

Output

```
[1, 3, [7, 4], 6]
```

```
>>> list2[2][0]=5 #Modifying the element at index 2,0
>>> list1
```

Output

```
[1, 3, [7, 4], 6]
```

```
>>> list2
```

Output

```
[1, 3, [5, 4], 6]
```

As you can see, this did not change anything for the original object.

2. Problems with Python Deep Copy

- It is possible that recursive objects cause a recursive loop; these are compound objects that directly or indirectly reference themselves.
- It is possible that a deep copy may copy too much.

To deal with these problems, deepcopy():

- Keeps a memo dictionary of objects is copied during the current copying pass.
- Allows user-defined classes to override the copying operation or the copied component set.

Python Shallow Copy

With a Shallow Copy in Python, we create a new object of which we recursively put copies of objects into the original.

In other words, we copy a reference of an object into another. Any changes we make to the copy do reflect in the original.

Let's implement this with Python. We'll use the copy() function.

```
>>> import copy
```

```
>>> list1=[1,3,[7,4],6]
>>> list2=copy.copy(list1) #Making a shallow copy
>>> list1
```

Output

```
[1, 3, [7, 4], 6]
```

```
>>> list2
```

Output

```
[1, 3, [7, 4], 6]
```

```
>>> list2[2][0]=5 #Modifying the element at index 2,0
>>> list1
```

Output

```
[1, 3, [5, 4], 6]
```

```
>>> list2
```

Output

```
[1, 3, [5, 4], 6]
```

It is apparent that making changes to a shallow copy does change the original object.

Shallow Copying Dictionaries in Python

Let's first create a dictionary in Shallow Copy in Python.

```
>>> dict1={'a':1,'b':2,'c':[1,2,3]}
```

Now, we'll make a copy of it.

```
>>> dict2=dict1.copy()
```

Finally, we'll append a new element to this.

```
>>> dict2['c'].append(7)
>>> dict1
```

Output

```
{'a': 1, 'b': 2, 'c': [1, 2, 3, 7]}
```

```
>>> dict2
```

Output

```
{'a': 1, 'b': 2, 'c': [1, 2, 3, 7]}
```

Other ways to create a shallow copy of a dictionary are dict(dict1) and copy.copy(dict1).

As you can see, altering the mutable in the copy changed the original too. To prevent this, we can do a deep copy in Python.

```
>>> import copy
>>> dict1={'a':1,'b':2,'c':[1,2,3]}
>>> dict2=copy.deepcopy(dict1)
>>> dict2['c'].append(7)
>>> dict1
```

Output

```
{'a': 1, 'b': 2, 'c': [1, 2, 3]}
```

```
>>> dict2
```

Output

```
{'a': 1, 'b': 2, 'c': [1, 2, 3, 7]}
```

So, this was all in Copy in Python. Hope you like our explanation of Shallow Copy and Deep Copy.

Python Interview Questions on Deep Copy and Shallow Copy

1. What is Copy in Python?
2. What does Copy() do in Python?
3. How do you make a Copy in Python?
4. What is difference between deep copy and shallow copy?

5. How does Deep Copy and Shallow Copy work in Python?

Conclusion

Hence, today, in this Copy in Python Tutorial, we discussed Shallow copy and Deep copy with Python.

Where shallow copy changes reflect in the original object, deep copy changes don’t.

Moreover, we understood a difference between Shallow Copy vs Deep Copy in Python. Also, we saw Shallow Copy Dictionaries.